

## Pipeline de instrucciones

### Manejo de Interrupciones

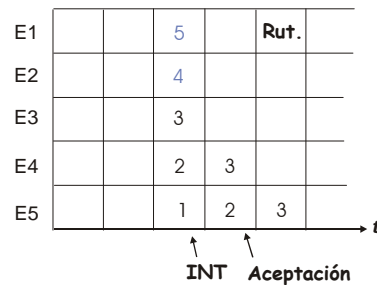
- Tipos:
- Síncronas
  - Asíncronas

**Asíncronas:** No están asociadas a ninguna instrucción.

Se atienden normalmente al final de la instrucción en ejecución.

➡ Permiten cierta flexibilidad.

Ej:



- 1 -

Aumento de prestaciones (10-11)

## Pipeline: Interrupciones

### Síncronas:

Se producen siempre en el mismo lugar, cada vez que se ejecuta el mismo programa con los mismos datos.

- Se dan y deben atenderse en "mitad" de una instrucción.
- Implican la detención y, en general, el reinicio de dicha instrucción.

El tratamiento de la mayoría de estas excepciones requiere:

- Salvar el **estado**.
- Ejecutar una **rutina de tratamiento**.
- **Restaurar** el estado y reiniciar.

### Interrupciones Precisas.

Las instrucciones que preceden a la que produjo la excepción se completan y las que le suceden se reinician.

➡ El mismo comportamiento que en un computador sin *pipeline*.

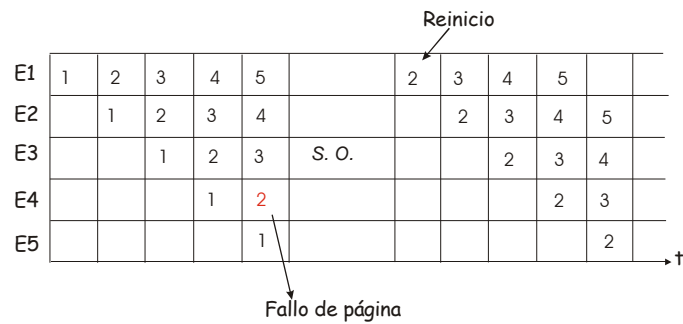
- 2 -

Aumento de prestaciones (10-11)

## Pipeline: Interrupciones

### Interrupciones síncronas.

**Ejemplo:** Fallo de página.



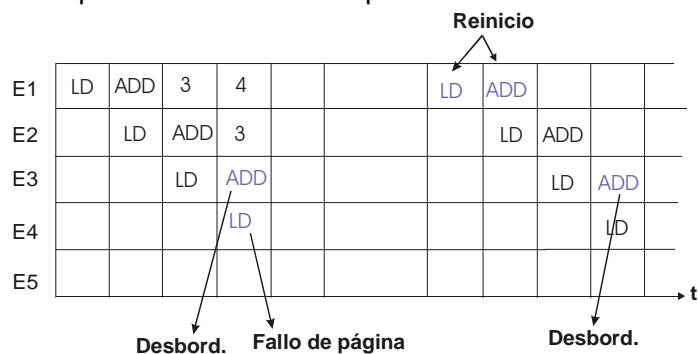
- 3 -

Aumento de prestaciones (10-11)

## Pipeline: Excepciones

**Problemas** que plantea el *pipeline* de instrucciones.

1. Pueden producirse **varias** excepciones **en el mismo ciclo**.



- 4 -

Aumento de prestaciones (10-11)



## Operaciones multiciclo

- Operaciones enteras complejas (mult, div).
- Operaciones de coma flotante.

Necesitan **más tiempo en la etapa EJ**.

¿Cómo abordar el *pipeline*, considerando estas operaciones?

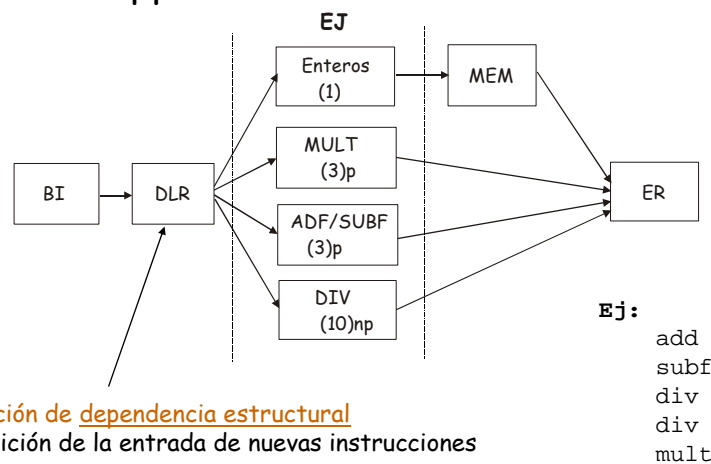
1. Prolongando la etapa EJ de estas operaciones varios ciclos.  
→ **No cambia el T.ciclo**
2. Alus especializadas vs. Alus multifunción
3. División de la labor de Ejecución en **varias unidades independientes**: enteros, coma flotante, ..., algunas con *pipeline*.

- 7 -

Aumento de prestaciones (10-11)

## Operaciones multiciclo

Estructura del pipeline:



Detección de dependencia estructural  
e inhibición de la entrada de nuevas instrucciones

- 8 -

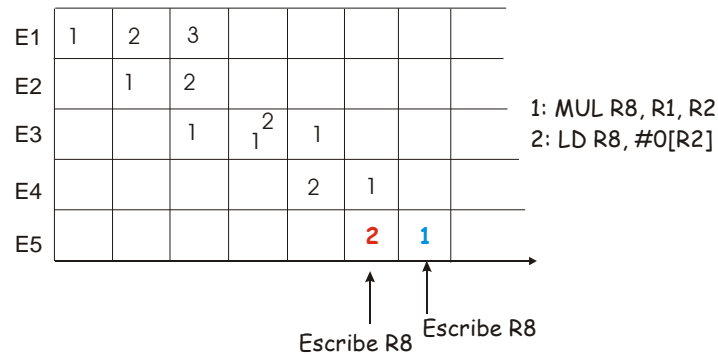
Aumento de prestaciones (10-11)



## Operaciones multicitlo

4) Pueden aparecer **dependencias** de datos **WAW**.

Ej:



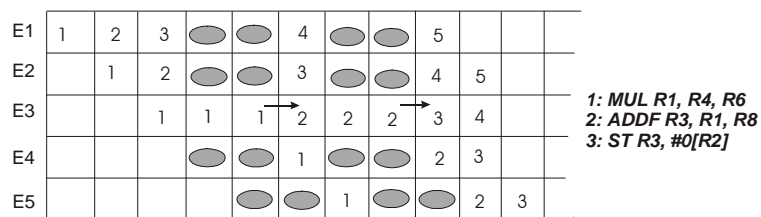
- 11 -

Aumento de prestaciones (10-11)

## Operaciones multicitlo

5) Los **parones** debidos a dependencias **RAW** serán **más frecuentes**, debido a la mayor latencia de algunas operaciones.

Ej:



Si las dependencias se detectan en E2, **antes de enviar una instrucción a E3** han de comprobarse:

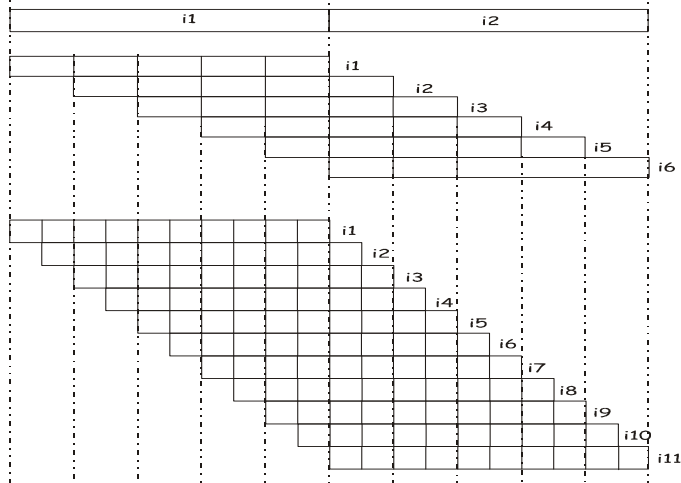
- » Las **dependencias estructurales**.
- » Las **dependencias RAW y WAW**.

- 12 -

Aumento de prestaciones (10-11)

## Superpipeline/Hiperpipeline

**Ganancia**<sub>ideal</sub> = n° de etapas. → Número elevado de etapas.



- 13 -

Aumento de prestaciones (10-11)

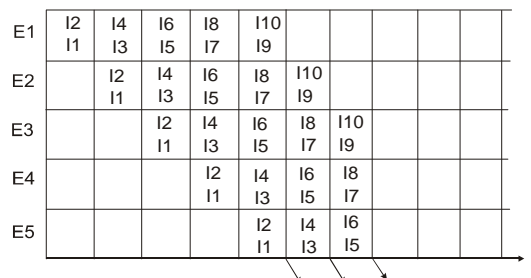
## Procesadores Superescalares

**OBJETIVO** :  $CPI_{ideal} < 1$

Para ello es necesario realizar **en un mismo ciclo**:

- *Fetch*
- Decodificación y lectura de operandos.
- Ejecución.
- Almacenamiento de resultados.

¡ De varias instrucciones ! **Múltiples pipelines en paralelo.**



2 instrucciones por ciclo

$CPI_{ideal} = 0,5$

- 14 -

Aumento de prestaciones (10-11)

## Procesadores Superescalares

- Posibilidad de "emitir" instrucciones **fuera de orden**.

➡ Mejora de prestaciones.

Ej:

```
div r3, r2, r4
addf r10, r3, r8    Dependencia RAW
subf r12, r8, r2
ld r5, #0(r20)      Instrucciones independientes
```

- Pueden aparecer **dependencias WAR**.

Ej:

```
ld r7, r5, #0(r1)
addf r6, r4, r8
div r4, r3, r6      Dependencia RAW
subf r3, r5, r1      WAR si subf finaliza antes que div
```

**WAR** y **WAW** son **falsas dependencias**.

Se resuelven mediante **Renombrado de registros** estático o dinámico.

- 15 -

Aumento de prestaciones (10-11)

## Procesadores Superescalares

*Ejemplo de renombrado de registros:*

```
r3 <- r3 op r5
r4 <- r3 op r6
r3 <- r5 op r1
r4 <- r8 op r1
```

RAW (from r3 <- r3 op r5 to r4 <- r3 op r6)

WAR (from r4 <- r3 op r6 to r3 <- r5 op r1)

WAW (from r3 <- r5 op r1 to r4 <- r8 op r1)

- 16 -

Aumento de prestaciones (10-11)



## Procesadores Vectoriales

### Aplicación:

Problemas que requieren mucho cálculo y pueden ser formulados en términos de vectores y matrices.

#### - Solución convencional:

```
for (i=1; i<100; i++)
    c[i] = a[i] + b[i]
```

#### - Solución vectorial:

```
c[I] = a[I] + b[I] (I=1,100)
```

### Modelos de Ejecución:

- Memoria-Memoria (Primeros vectoriales)  
Alto ancho de banda de la memoria (CYBER: 512 bits)
- Registro-Registro.  
Registros de alta capacidad (CRAY: 64 elementos de 64 bits).

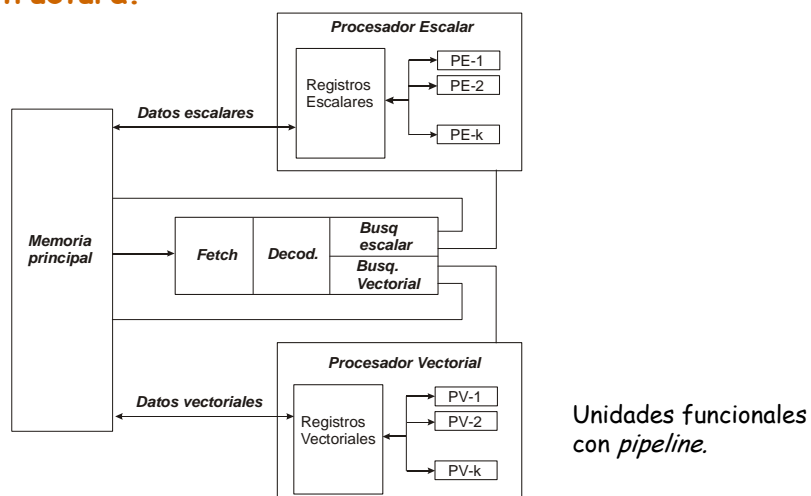
### Uso intensivo de *pipeline* aritmético

- 17 -

Aumento de prestaciones (10-11)

## Procesadores Vectoriales

### Estructura.

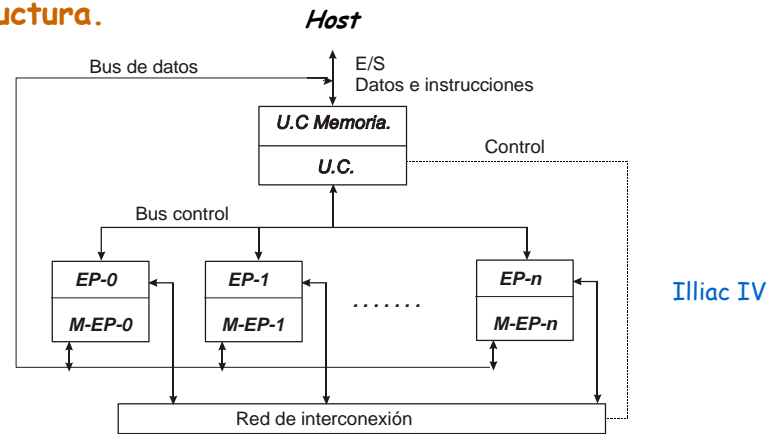


- 18 -

Aumento de prestaciones (10-11)

## Procesadores SIMD (Matriciales)

### Estructura.



La U.C. lanza la misma instrucción a todos los EP, que la ejecutan sobre sus datos locales (síncronos).

- 19 -

Aumento de prestaciones (10-11)

## Sistemas MIMD

### Características globales:

- Dos o más procesadores con la misma capacidad de cálculo.
- Funcionamiento asíncrono.
- Cada procesador tiene su propia U. Control.
- Grado de acoplamiento.(Grado de interacción entre procesadores):
  - Débil (Multicomputadores).
  - Moderado (NUMA)
  - Fuerte (UMA)
- Escalabilidad:
  - Escalables.
  - No escalables.
- Simetría.
  - Simétricos
  - Asimétricos (procesadores dedicados para E/S: *attached*).

- 20 -

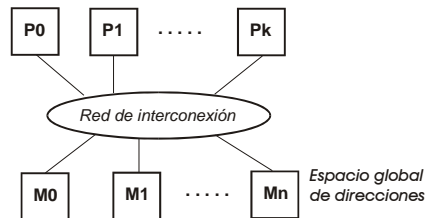
Aumento de prestaciones (10-11)

## Clasificación de MIMDs

### MULTIPROCESADORES

Memoria compartida (Lógica)

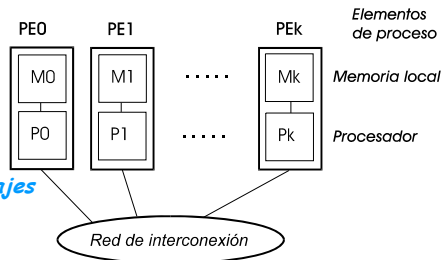
*Comunicación a través de memoria*



### MULTICOMPUTADORES

Memoria distribuida

*Comunicación mediante paso de mensajes*



- 21 -

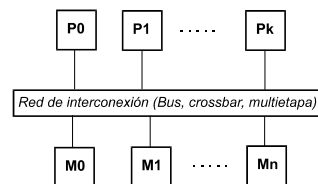
Aumento de prestaciones (10-11)

## Multiprocesadores

- Un solo espacio de direcciones lógico visible a todos los procesadores.
- Dependiendo de la organización física de la memoria:

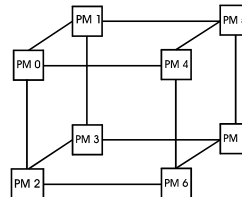
### UMA.

- Memoria física y lógica compartida
- Acceso uniforme a memoria.



### NUMA

- Memoria lógica compartida
- Acceso no uniforme a memoria.



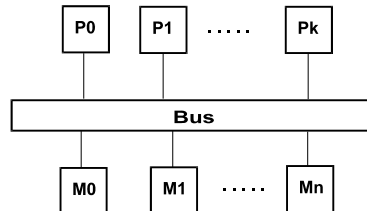
- 22 -

Aumento de prestaciones (10-11)

## Multiprocesadores

### 1ª GENERACIÓN:

- UMA's con bus compartido.



- Problemas de latencia de memoria:
  - Acceso de varios procesadores al mismo módulo de memoria.
  - Retrasos introducidos por la red de interconexión y conflictos en la propia red.

Cuanto mayor es el nº de procesadores, mayor será la latencia media → **Baja escalabilidad** (16-64 procesadores)

- 23 -

Aumento de prestaciones (10-11)

## Multiprocesadores UMA: Ejemplo

- CPU's de 32 bits y 1000 MIPS
- Arquitectura RISC
- Nº de CPU's = 20
- El 30% de las instrucciones son Load o Store: 70% Load y 30% Store

**Nº de Accesos** a memoria por segundo y por CPU:

- *Fetch* de instrucción: 1.000.000.000
  - Búsqueda de operando (*load*):  $1.000.000.000 \times 0,3 \times 0,7$
  - Escritura de resultado (*store*):  $1.000.000.000 \times 0,3 \times 0,3$
- 
- 1.300.000.000 peticiones/s

**Ancho de banda necesario** en la memoria compartida:

$$20 \times 1.300.000.000 \text{ palabras/s} = 26.000.000.000 \text{ palabras/s}$$

**T. Acceso necesario:**

**¡ 0,038 ns !**

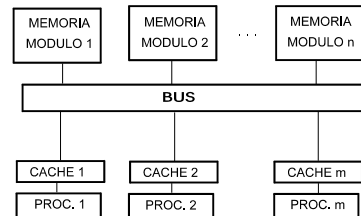
- 24 -

Aumento de prestaciones (10-11)

## Multiprocesadores UMA

Para reducir tráfico, contención y latencia:

### Uso de memorias cache locales



### PROBLEMA: Coherencia de caches

- Por compartición de datos "modificables".
- Por migración de procesos.

### SOLUCIONES:

- Usar la CACHE sólo para código (*read-only*) y datos privados.
- Algoritmos de coherencia: Protocolos *snoopy* (ej: MESI)

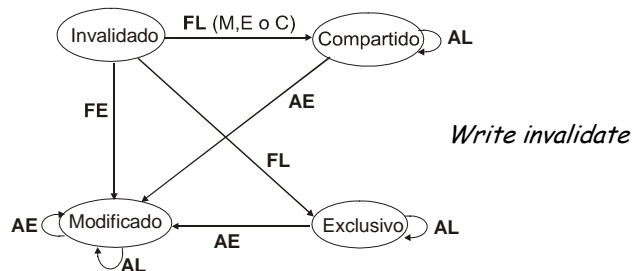
- 25 -

Aumento de prestaciones (10-11)

## Protocolos *snoopy*

Dotar a las Mca de la habilidad de "escuchar" las peticiones que van por el bus y actuar en consecuencia.

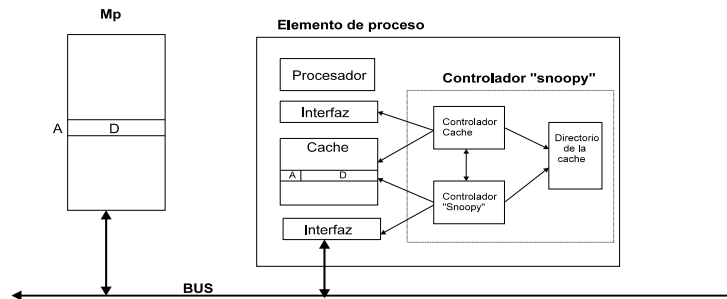
- Caches con *write-through*:
  - Invalidación (*write invalidate*)
  - Actualización (*write update*)
- Caches con *copy-back*: protocolos más complejos:
  - "write once" → MESI



- 26 -

Aumento de prestaciones (10-11)

## Multiprocesadores UMA



### Ejemplos:

|                         |    |              |        |          |
|-------------------------|----|--------------|--------|----------|
| Compaq Proliant 5000    | 4  | Pentium Pro  | 200MHz | 2.048MB  |
| Digital AlphaServer8400 | 12 | Alpha21164   | 440MHz | 28.672MB |
| HP9000 K460             | 4  | PA-8000      | 180MHz | 4.096MB  |
| IBM RS/6000 R40         | 8  | PowerPC 604  | 112MHz | 2.048MB  |
| SGI Power Challenge     | 36 | MIPS R10000  | 195MHz | 16.384MB |
| Sun Enterprise 6000     | 30 | UltraSparc 1 | 167MHz | 30.720MB |

- 27 -

Aumento de prestaciones (10-11)

## Multiprocesadores

### 2ª GENERACIÓN: NUMAs.

- Distribución física de la memoria entre los procesadores.
- Redes de interconexión más complejas que bus.
- Alta escalabilidad y no coherencia de caches.

### 3ª GENERACIÓN: CC-NUMAs.

- Grandes caches locales y utilización de protocolos de coherencia no *snoopy* (directorio).

|                      |      |             |        |           |       |
|----------------------|------|-------------|--------|-----------|-------|
| Cray T3E (NUMA)      | 2048 | Alpha 21164 | 450MHz | 524.288MB | 4-SMP |
| Convex Exemplar      | 64   | PA8000      | 180MHz | 65.536MB  | 2-SMP |
| Sequent NUMA-Q       | 32   | Pent.Pro    | 200MHz | 131.072MB | 4-SMP |
| SGI Origin2000       | 128  | MR10000     | 195MHz | 131.072MB | 2-SMP |
| Sun Enterprise 10000 | 64   | UltraSparc1 | 250MHz | 65.536MB  | 4-SMP |

- 28 -

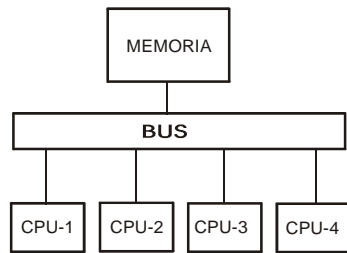
Aumento de prestaciones (10-11)

## Redes de interconexión

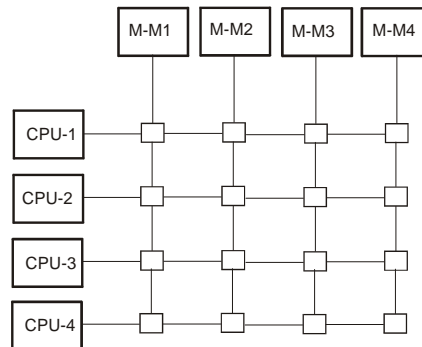
Diversas topologías.

**DINÁMICAS:** Utilizadas en multiprocesadores UMA.

**Bus compartido**



**Crossbar**



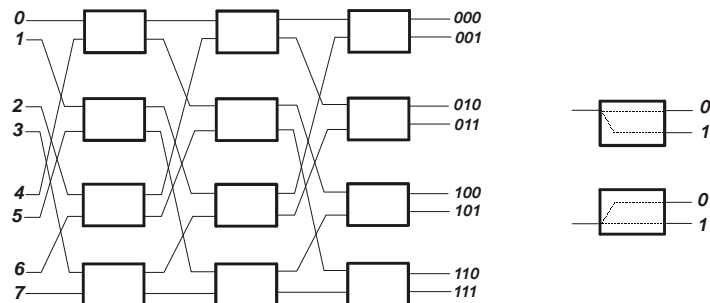
- 29 -

Aumento de prestaciones (10-11)

## Redes de interconexión

**DINÁMICAS:**

**Red multietapa**



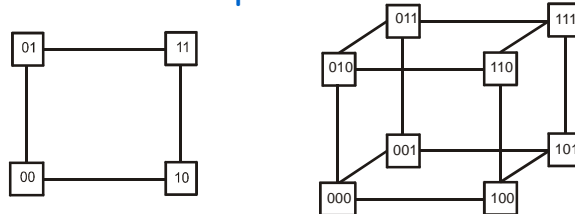
- 30 -

Aumento de prestaciones (10-11)

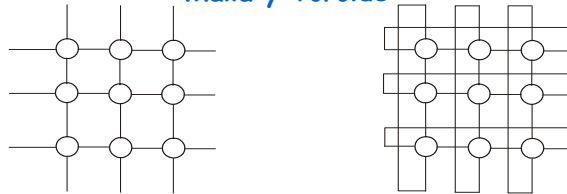
## Redes de interconexión

**ESTÁTICAS:** Utilizadas en multiprocesadores NUMA

### Red hipercubo



### Malla y Toroide



- 31 -

Aumento de prestaciones (10-11)